Programação Orientada a Objetos

HERANÇA

Profa. Nadia Félix

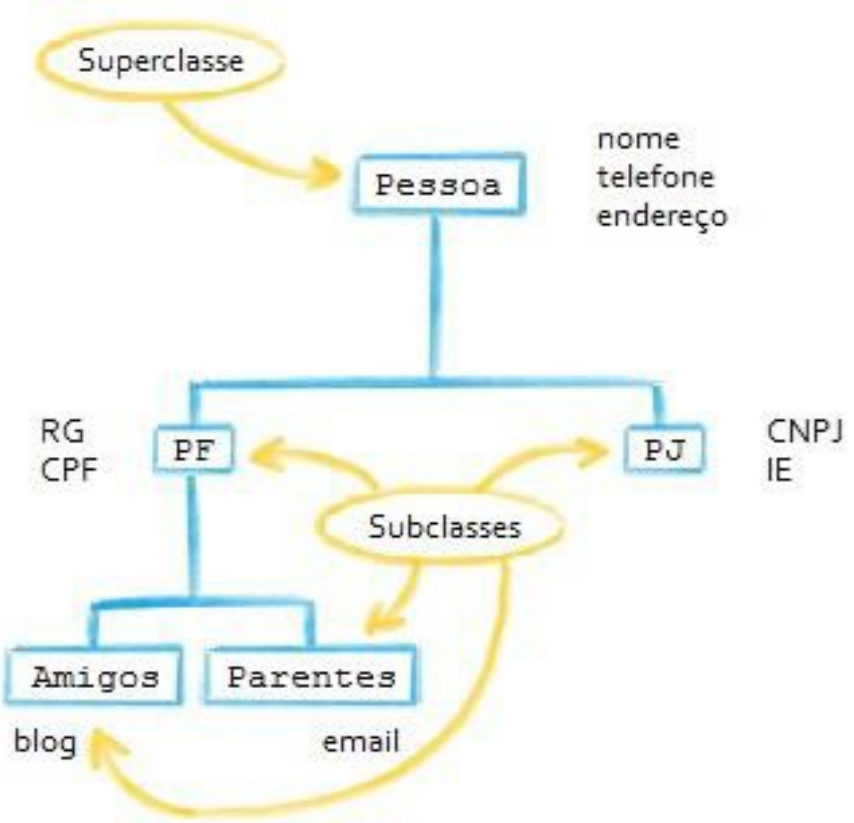
nadia.felix@ufg.br

Prof. Dirson Santos de Campos

dirson_campos@ufg.br

Programação Orientada a Objetos

- HERANÇA
- Reutilização de Código



Programação Orientada a Objetos

Herança

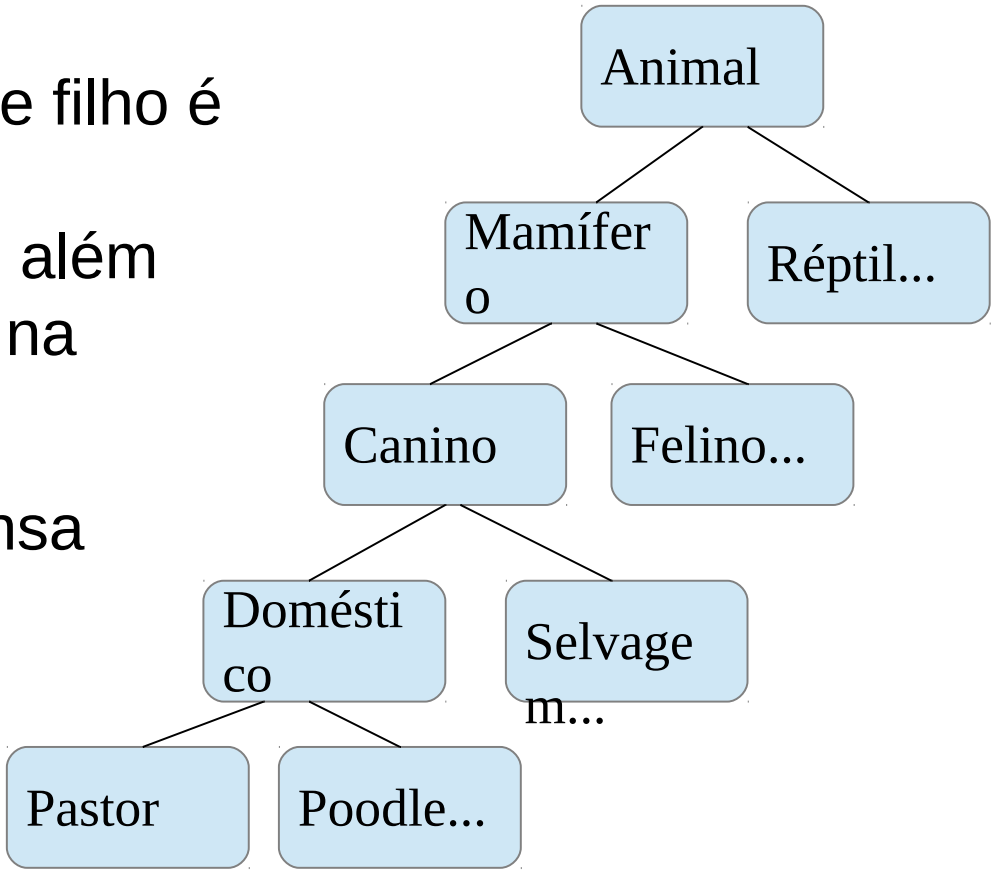
- Mecanismo de reutilização de software onde uma nova classe é criada absorvendo membros de uma classe existente e aprimorada com capacidades novas ou modificadas;

- Permite que elementos mais específicos incorporem a estrutura e o comportamento de elementos mais genéricos;

Programação Orientada a Objetos

Herança (ou generalização)

- Quando um objeto da classe filho é criado ele herda todas as propriedades da classe pai, além das propriedades definidas na própria classe filho;
- O homem naturalmente pensa dessa forma ...



Programação Orientada a Objetos

É um tipo de ...

- Um objeto de uma subclasse (classe filha) **é um tipo de** objeto da superclasse (classe pai);

Beagle  É um tipo de Cachorro

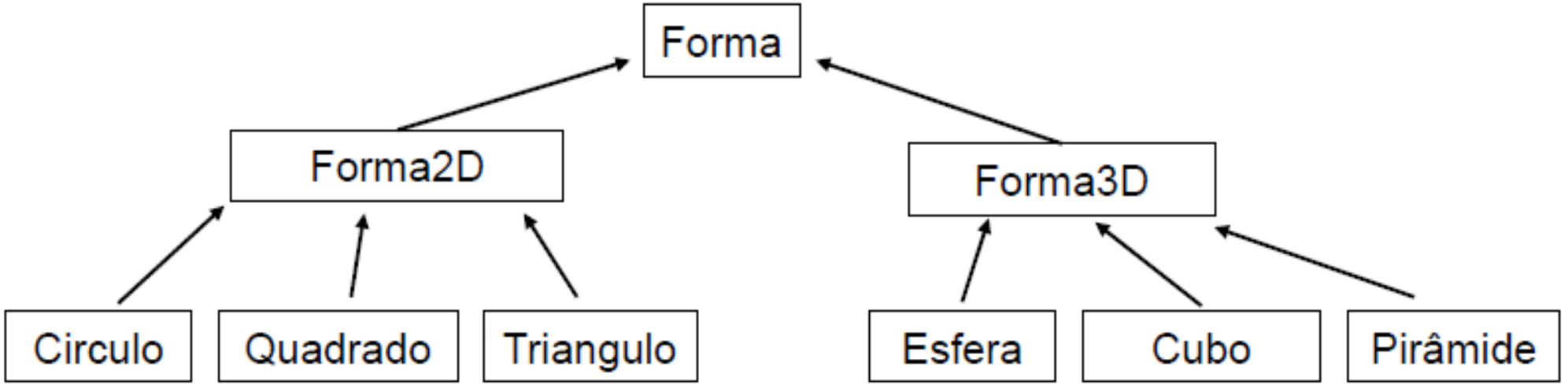
É um tipo de veículo

Programação Orientada a Objetos

Herança (cont.)

- Frequentemente um objeto de uma determinada classe também ***é um*** objeto de outra classe.

- Este tipo de relação normalmente é hierarquizada ...



Programação Orientada a Objetos

Superclasses X Subclasses

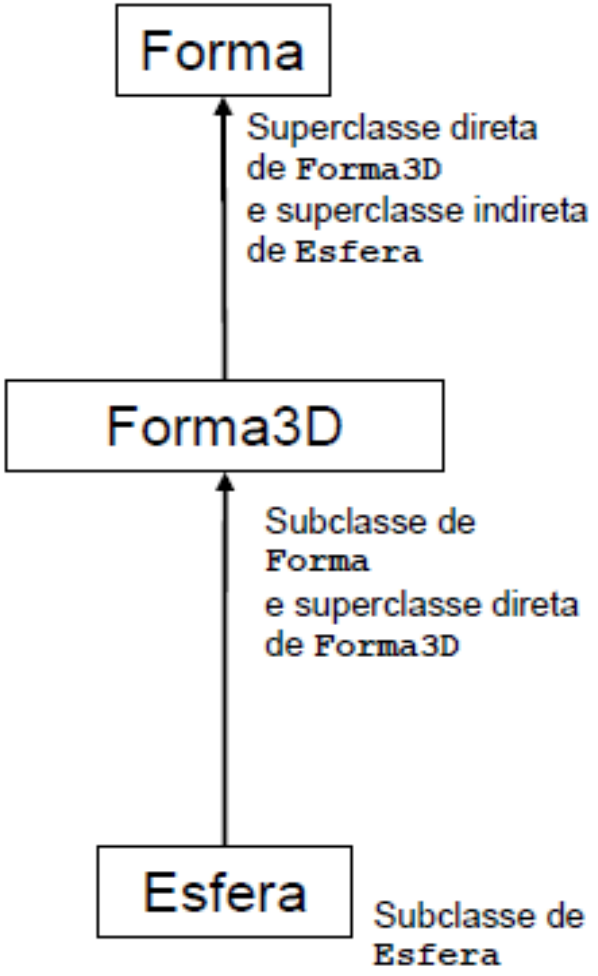
- Quando especificamos uma classe, ao invés de começar do zero, declarando atributos e métodos que talvez já existam em outra classe podemos designar a nova classe a herdar o comportamento e as ações de uma classe já existente;

- A classe existente é chamada de **superclasse** e a nova classe de **subclasse**.

Programação Orientada a Objetos

Superclasses X Subclasses

- Superclasse tendem a ser mais gerais enquanto que subclasses, mais específicas;
- Toda subclasse pode vir a tornar-se uma superclasse para futuras subclasses;
- A superclasse direta é aquela a partir do qual a subclasse herda explicitamente, uma superclasse indireta é qualquer superclasse acima da classe direta na hierarquia de classes.



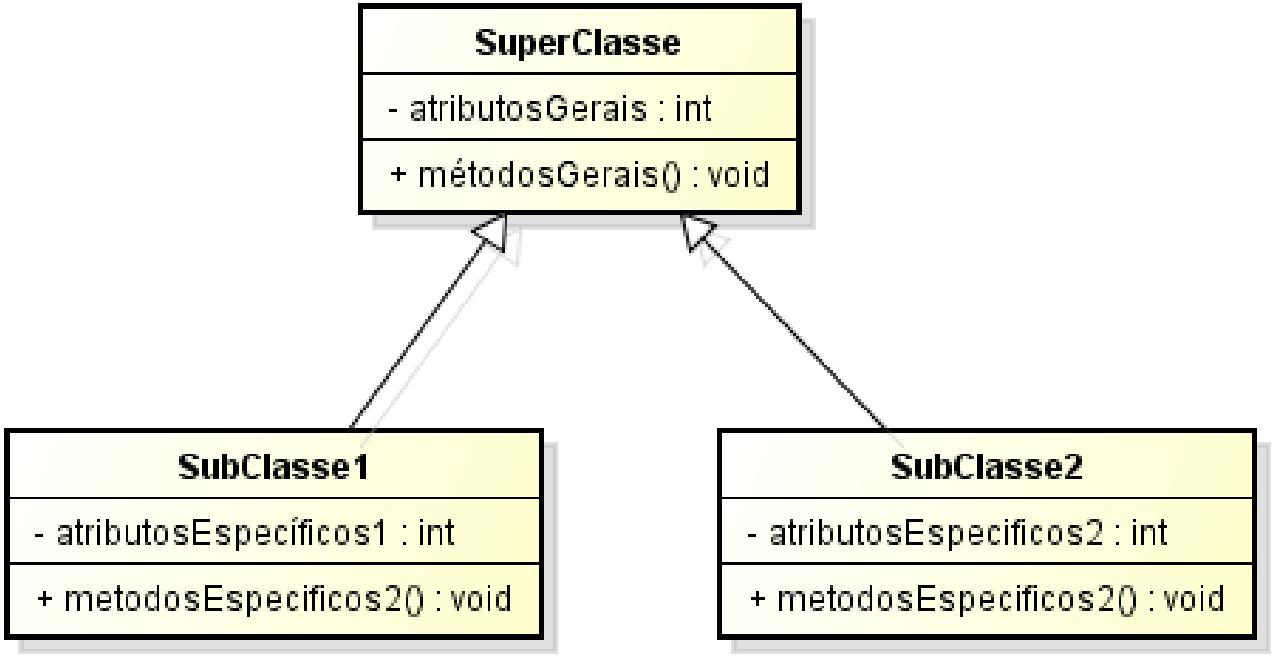
Programação Orientada a Objetos

Como identificar e modelar a herança ...

- Identificar as entidades importantes do contexto;
- Identificar as características (dados) e comportamentos (operações) de cada uma;
- Identificar características e comportamentos comuns (gerais) nas entidades;
- Identificar características e comportamentos específicos em cada entidades;
- Agrupar características e comportamentos comuns em uma superclasse (classe pai);
- Manter características e comportamentos específicos em cada classe;

Programação Orientada a Objetos

Herança - Representação UML



Programação Orientada a Objetos

Exemplo de atores em uma Loja

Uma loja deseja modelar e desenvolver um sistema e para isso ele identificou algumas entidades/atores importantes nas suas operações diárias: clientes, fornecedores e funcionários.
Verificou-se que todos podiam ser considerados **Pessoas** pois têm várias características comuns, porém alguns têm características especiais que também devem ser consideradas e representadas no modelo.
Para facilitar a modelagem dos dados e a implementação do sistema, foram utilizados os conceitos de herança para criar as classes, tentando reaproveitar o máximo de código possível.

Programação Orientada a Objetos

Exemplo de atores em uma Loja

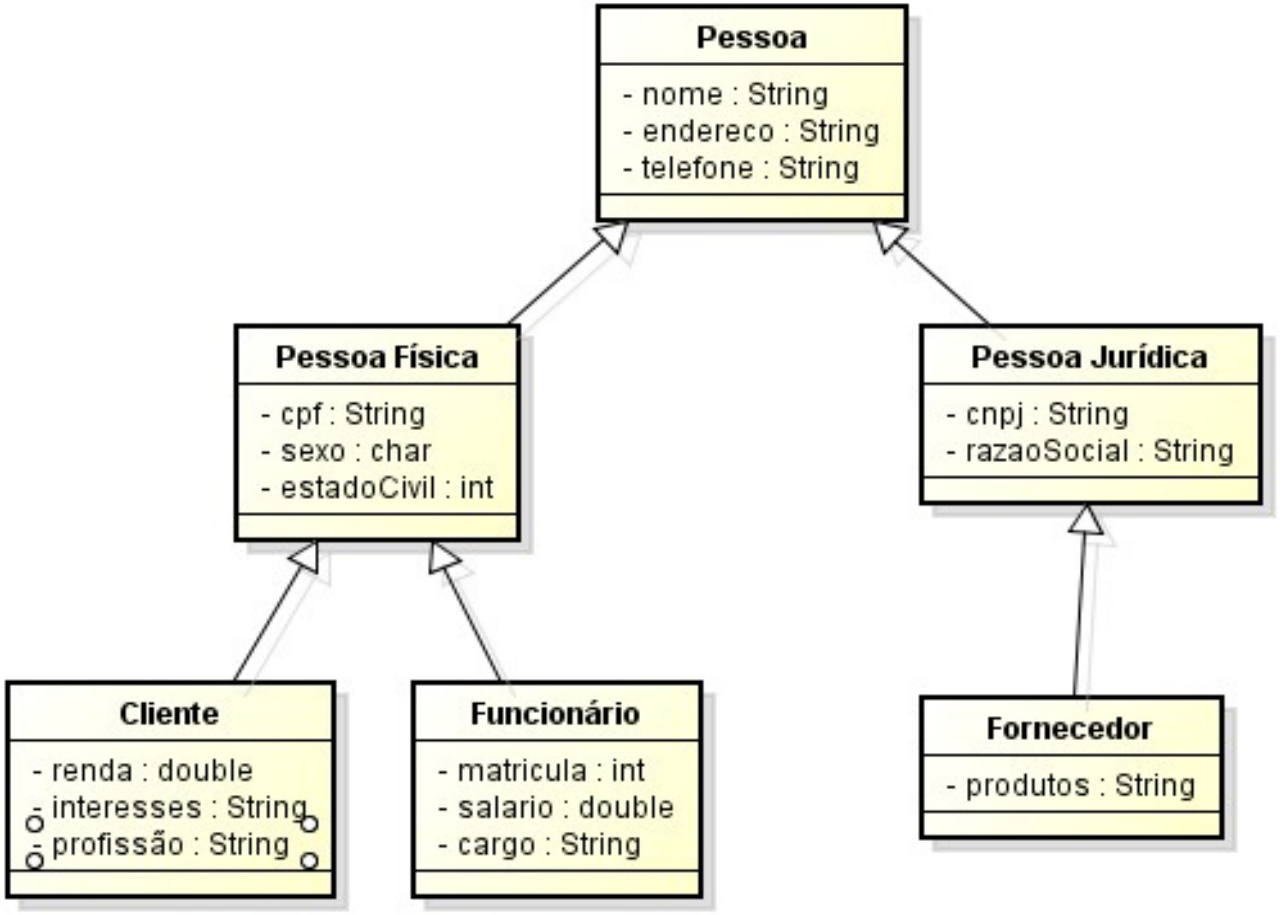
Como você faria isso considerando as seguintes características:

- Um **cliente** é uma pessoa física que para ser cadastrado na loja precisa informar seu nome, cpf, endereço, telefone, sexo, estado civil, renda, interesses e profissão.
- Um **funcionário** é uma pessoa física e a loja guarda as seguintes informações sobre ele: nome, endereço, telefone, cpf, matrícula, cargo e salário
- Um **fornecedor** é uma entidade importante na loja e para ser cadastrado precisa informar o seu nome, endereço e telefone, cnpj, razão social e os produtos que ele fornece.

Tente gerar um modelo com essas informações

Programação Orientada a Objetos

Exemplo de atores em uma Loja



Programação Orientada a Objetos

Herança em Java

- public class PessoaFisica extends Pessoa {
- }
- A palavra reservada **extends** indica que a classe a ser especificada herda de uma outra classe;
- Na linguagem Java a hierarquia de classes inicia com a classe Object (do pacote java.lang), sendo assim toda classe Java é descendente em algum grau da classe Object.
- Uma sub-classe tem acesso aos atributos e métodos definidos com visibilidade **public** e **protected**, mas **não** **private**.

Programação Orientada a Objetos

Herança em Java

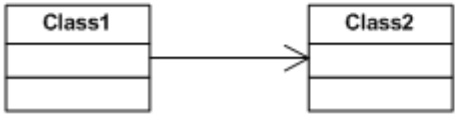
```
public class Pessoa {
  – protected String nome;
  – protected String endereco;
  – protected String telefone;
}
public class PessoaFisica extends Pessoa {
  – protected String cpf;
  – protected char sexo;
  – protected int estadoCivil;
  – }
public class PessoaJuridica extends Pessoa {
  – protected String cnpj;
  – protected String razaoSocial;
}
```

```
public class Cliente extends PessoaFisica {
    protected double renda;
    protected String interesses;
    protected String profissao;
}

public class Funcionario extends PessoaFisica {
    protected int matricula;
    protected String cargo;
    protected double salario;
}

public class Fornecedor extends PessoaJuridica {
    protected String produtos;
}
```

Associação

- A associação entre dois objetos ocorre quando eles são completamente independentes entre si mas eventualmente estão relacionados. Ela pode ser considerada uma relação de muitos para muitos. Não há propriedade (*ownership*) nem dependência entre eles. A relação é eventual.
  - Um exemplo é a relação entre um professor e alunos. Um aluno pode ter vários professores e um professor pode ter vários alunos. Um não depende do outro para existir. Professores podem existir sem alunos e e alunos podem existir sem professores (pelo menos em requisitos normais).

```
class Cliente {
    var contatos = new List<Contato>();
    AdicionarContato(Contato contato) {
        contatos.Add(contato); //este contato independe deste cliente
    }
}
```



Agregação

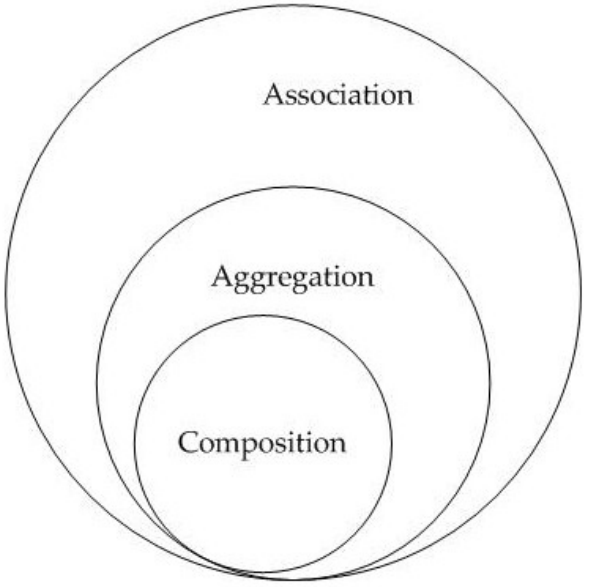
- A agregação não deixa de ser uma associação mas existe uma exclusividade e determinados objetos só podem se relacionar a um objeto específico.
- É uma relação de um para muitos.
- Um objeto é proprietário de outros mas não há dependência, então ambos podem existir mesmo que a relação não se estabeleça.

  Um exemplo é a relação entre os professores e os departamentos. Departamentos podem ter vários professores. E o professor só pode estar vinculado a um único departamento. Mas eles são independentes. Um professor pode existir sem vínculo com um departamento e este não depende de professores para existir. Outro exemplo:

Composição

- A composição é uma agregação que possui dependência entre os objetos, ou seja, se o objeto principal for destruído, os objetos que o compõe não podem existir mais. Há a chamada relação de morte.
- Um exemplo é a relação entre uma a universidade e os departamentos. Além da universidade possuir vários departamentos, eles só podem existir se a universidade existir. Há uma dependência.

Os termos possuem esta relação:



Sumarizando

- Em associação não temos dono. os objetos têm tempo de vida próprios. E os objetos filhos são independentes, ou seja existe qualquer tipo de relação entre os objetos e estes podem ser chamados independentemente uns a partir dos outros.

- Em agregação temos apenas um único dono. Os objetos tem tempo de vida independente. E os objetos filhos pertencem a um único parent. Ou seja existe uma espécie de ligação do tipo unilateral. Apesar de os objetos ainda serem minimamente independentes.

- Composição é idêntica a agregação mas o tempo de vida é o do dono. Ou seja, se o dono for parado, os outros também o são.

Herança X Composição

- Herança é a capacidade de uma classe herdar propriedades e comportamento de uma classe pai, estendendo-a.
- Composição é a capacidade de uma classe de conter objetos de diferentes classes como dados de membro.

Herança e Agregação

- Herança: "é um"
- Agregação: "tem um". Mostra que o objeto agregador possui um dos objetos agregados.

Programação Orientada a Objetos

Herança X Associação/Composição/Agregação

"é um" **X** "contém um"

- A composição é uma outra forma de reaproveitarmos classes (também é conhecido por delegação);
- Consiste em criar novas classes incluindo nelas atributos da classe que se quer reaproveitar.
- Para que os métodos da classe base possam ser executados, escrevemos métodos correspondentes na classe nova que chama os da classe base, delegando a execução dos métodos ...

Programação Orientada a Objetos

Herança X Associação/Composição/Agregação

- A classe Aluno contém na sua relação de atributos uma instância de Curso que tem uma instância de Universidade.

```
public class Aluno {
  String nome;
  int matricula;
  Curso curso;
  Aluno(int matricula, String nome)
{
    this.nome = nome;
    this.matricula= matricula;
 }
}
public class Curso {
  String nome;
  int codigo;
  Universidade universidade;
  Curso (int codigo,String nome) {
    this.nome = nome;
    this.codigo = codigo;
 }
}
```

```
public class Universidade {
  String nome;
  String sigla;
  Universidade(String nome, String sigla)
{
    this.nome = nome;
    this.sigla= sigla;
  }
}
```

Programação Orientada a Objetos

Sobrescrevendo Métodos

- Modificação de um método da superclasse na subclasse;
- Toda vez que um método que já existe na superclasse é redeclarado na subclasse ele oculta o método da superclasse;

```
class Animal {

     protected String nome;

     public String fazerBarulho() {
          return "Barulho de um Animal";
     }
}

class Cao extends Animal {

     protected String raca;

     public String fazerBarulho() {
          return "Barulho do cão - Latido";
     }
}
```

```
class Gato extends Animal {

     protected int fiosBigode;

     public String fazerBarulho() {
          return "Barulho do gato - Miau";
     }
}


public class ReinoAnimal {

     public static void main(String[] args) {
          Gato g = new Gato();
          g.nome = "Fofinho";
          System.out.println(g.fazerBarulho());
     }
}
```

Programação Orientada a Objetos

Referência **`super`**

- Permite às subclasses acessarem métodos das superclasses;
- A palavra reservada **`super`** é similar a **`this`**, porém atua como referência para o objeto corrente interpretado como uma instância da superclasse;
- Construtores da superclasse são chamados simplesmente pela palavra **`super`** (seguida de eventuais argumentos), demais métodos da superclasse são chamados pela palavra **`super`** seguida do sinal de ponto e o nome do método.
  - Construtores de superclasses só podem ser chamados a partir de construtores de subclasses, e devem obrigatoriamente ser a primeira linha;
  - Somente métodos da superclasse imediata podem ser acessados.

Programação Orientada a Objetos

Referência **super (cont..)**

```
class Animal {

     protected String nome;

     public String fazerBarulho() {
          return "Barulho de um Animal";
     }
}

class Cao extends Animal {

     protected String raca;

     public String fazerBarulho() {
          return "Barulho do cão - Latido";
     }
}
```

```
class Gato extends Animal {

     protected int fiosBigode;

     public String fazerBarulho() {
          return super.fazerBarulho() + " gato - Miau";
     }
}


public class ReinoAnimal {

     public static void main(String[] args) {
          Gato g = new Gato();
          g.nome = "Fofinho";
          System.out.println(g.fazerBarulho());
     }
}
```

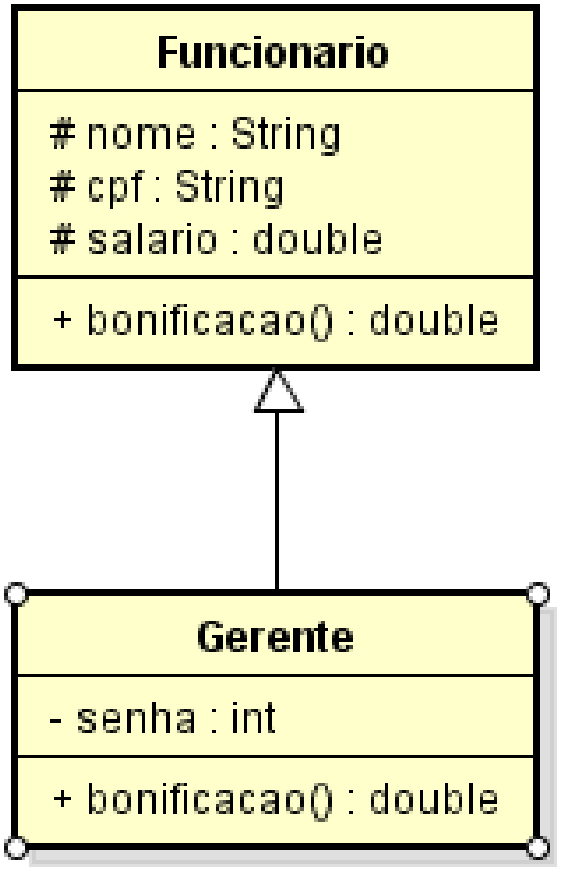
Programação Orientada a Objetos

Exemplo:

- Uma classe Funcionário tem nome, cpf e salário. Tem também um método que calcula bonificação de 10% do salário.
- Todo Gerente, é um Funcionário e como tem acesso restrito a alguns lugares, ele tem uma senha.
- O cálculo da bonificação do Gerente é diferente, sendo 15% do salário.

Programação Orientada a Objetos

Exemplo:



Programação Orientada a Objetos

Exemplo:

```
public class Funcionario {
    protected String nome;
    protected String cpf;
    protected double salario;

public double bonificacao(){
    double b = salario * 0.10;
    return b;
}

public double getSalario() {
    return salario;
}

public void setSalario(double
salario) {
    this.salario = salario;
}

}
```

```
public class Gerente extends
Funcionario{
    private int senha;

public double bonificacao(){
    double b = salario * 0.15;
    return b;
}

}
```

```
public class TestaHeranca {

public static void main(String [] args){
    Gerente g = new Gerente();
    g.setSalario(3000);
    System.out.println("A bonificacao é:
" +     g.bonificacao());
}

}
```

Programação Orientada a Objetos

Métodos construtores nas subclasses

- O construtor de uma subclasse **sempre** chama o construtor de sua superclasse, mesmo que a chamada não seja explícita.

- Se a chamada não for explícita (através da palavra-chave **`super`**) o construtor da subclasse tentará chamar o construtor vazio (sem argumentos) da superclasse – e se ele não estiver definido, ocorrerá um erro de compilação;

- Se uma classe não possui um construtor vazio (sem argumentos) e possui um construtor com argumentos, as classes herdeiras deverão obrigatoriamente chamar o construtor com argumentos da classe ancestral (este é um tipo de erro que geralmente causa muita confusão).

Programação Orientada a Objetos

```
public class Funcionario {
    protected String nome;
    protected String cpf;
    protected double salario;

Funcionario(String nome,String cpf){
    This.nome = nome;
    This.cpf = cpf;
}
...
}
```

```
public class Gerente extends
Funcionario{
    private int senha;

Gerente (String nome,String cpf, int
senha){
    super(nome,cpf);
    This.senha = senha;
}


}
```

```
public class TestaHeranca {

public static void main(String [] args){
    Funcionario f = new Funcionario(“Jose”,”23435678999”);
    Gerente g = new Gerente(“Paulo”,”9949595595”,3344);
    System.out.println("Funcionario: " + f.getNome());
    System.out.println("Gerente: " + g.getNome());
}
}
```